Randomness, Interactive Proofs and Zero-Knowledge

(a survey)

Oded Goldreich
Computer Science Dept.
Technion, Haifa, Israel

ABSTRACT

Recent approaches to the notions of randomness and proofs are surveyed. The new notions differ from the traditional ones in being subjective to the capabilities of the observer rather than reflecting "ideal" entities. The new notion of randomness regards probability distributions as equal if they cannot be told apart by efficioent procedures. This notion is constructive and is suited for many applications. The new notion of a proof allows the introduction of the notion of *zero-knowledge proofs*: convincing arguments which yield nothing but the validity of the assertion.

The new approaches to randomness and proofs are based on basic concepts and results from the theory of resource-bounded computation. In order to make the survey as accessible as possible, we have presented elements of the theory of resource bounded computation (but only to the extent required for the description of the new approaches).

This survey is not intended to provide an account of the more traditional approaches to randomness (e.g. Kolmogorov Complexity) and proofs (i.e. traditional logic systems). Whenever these approaches are described it is only in order to confront them with the new approaches.

Appears in *The Universal Turing Machine: A Half-Century Survey*, R. Herken ed., Kammerer/Unverzagt Verlag and Oxford University Press, 1987.

1. INTRODUCTION

When talking about a universal Turing machine, it is implicitly postulated that the input-output relation of the machine (i.e. the function it computes) is the only aspect of the machine that we are interested in. This postulate is naturally imposed upon us, when viewing Turing machines as *means* for defining whatever "can be automatically computed". However, when viewing Turing machines (i.e. the computation devices themselves) as the *subject* of an investigation, concentration on their input-output relation should be considered a behaviouristic approach.

Recent research treads in computational complexity (i.e. the theory of resource-bounded computation) have even a stronger behaviouristic flavour. In these works, one does not consider the input-output relation of a Turing machine, but rather its effect on an (arbitrary) observer (i.e. another Turing machine) with certain natural resource bounds.

Typically, the results we will survey state that the input-output behaviour of "ontologically" different machines, look the same to any suitably resource-bounded observer. For example, pseudorandom sequences are defined to be indistinguishable in polynomial-time from truly random sequences, although the first may be generated using substantially fewer coin tosses. Another example is the zero-knowledge proofs, which have the remarkable property of being both convincing and "reproducible" in a slightly weaker sense by parties which do not have a "real proof".

2. BACKGROUND : RESOURCE – BOUNDED COMPUTATION

We begin this section by recalling the definitions of **P**, **NP** and **BPP** – the complexity classes corresponding to deterministic, non-deterministic and probabilistic polynomial-time computations. We continue by presenting the definition of one-way functions, which plays a central role in the construction of pseudorandom generators and in the general results concerning zero-knowledge proofs.

The theory of resource bounded computations is developed in terms of asymptotic behaviour. Typically, we will consider the number of steps taken by a Turing machine as a function of its input length, bound this function from above by a "smoother" function, and ask whether the bound is (or can be) a polynomial. This convention allows us to disregard special (short) inputs on which the machine may behave exceptionally good, and helps us concentrate on the "typical" behaviour of the machine.

2.1. Deterministic Polynomial-Time Computations

Traditionally in computer science, deterministic polynomial-time computations are associated with efficient computations. (Throughout the article, a *polynomial-time computation* means a computations in which the number of elementary computing steps is bounded by a polynomial in the length of the input.) This association stems from the acceptability of determinististic computing steps and polynomial-time computations as feasible in practice. The preference of deterministic computing steps (over non-deterministic ones) is self evident. Polynomial-time computations are advocated as efficient due to the moderate growing rate of polynomials and due to the correspondence between problems which are *known* to have "practically efficient" solutions and those *known* to have polynomial-time solutions.

Deterministic polynomial-time computations are captured by the complexity class **P** (*P* stands for **P**olynomial). The complexity class **P** is defined as the set of languages satisfying for each $L \in \mathbf{P}$ there exists a Turing machine M and a polynomial $p(\cdot)$ such that the following two conditions hold:

1) On input a bit string x ($x \in \{0,1\}^*$), machine M halts after at most $p(|x|)$ steps, where $|x|$ is the length of the string x.

2) On input x, machine M halts in an *accepting state* if and only if $x \in L$. (Otherwise it halts in a "rejecting state".)

2.2. Non-deterministic Polynomial-Time Computations

Another interesting complexity class is the set of languages recognizable by non-deterministic [1] polynomial-time Turing machines denoted **NP** (*NP* stands for **N**on-deterministic **P**olynomial-time). Namely, a language L is in **NP** if there exists a non-deterministic machine M and a polynomial $p(\cdot)$ satisfying:

1) A non-deterministic Turing machine has a transition function which goes from the current local configuration to a (finite) *set of possible* local configurations, rather than going from the current local configuration to the next local configuration.

1) On input $x \in \{0,1\}^*$, machine M halts after at most $p(|x|)$ steps.

2) On input x, there *exists* a computation of M halting in an accepting state if and only if $x \in L$. (Otherwise *all* computations halt in a rejecting state.)

Equivalently, a language L is in **NP**, if there exists a Boolean predicate $P_L:\{0,1\}^*\times\{0,1\}^*\rightarrow\{0,1\}$ such that P_L is computable in (deterministic) polynomial-time and $x \in L$ if and only if there exists a y of length polynomial in |x| satisfying $P_L(x,y)=1$. (The string y can be thought of as encoding the non-deterministic choices of machine M above.) Thus, **NP** consists of the set of languages for which there exist short proofs of membership that can be efficiently verified. Both classes (**P** and **NP**) are defined in terms of decision problems (i.e. "is x in the set S?"). Equivalent formulations in terms of search problems (i.e. "given x find y such that (x,y) are in the relation R") are obvious. **P** can be viewed as the class of search problems for which a solution (i.e. a string satisfying the relation with the input) can be found in (deterministic) polynomial-time; while **NP** can be viewed as the class of problems for which a solution, once found, can be verified in (deterministic) polynomial-time.

It is widely believed that **P** ≠ **NP**. Settling this conjecture is certainly the most intriguing open problem in Theoretical Computer Science. If true, **P** ≠ **NP** means that there are search problems for which verifying the validity of a solution is substantially easier than coming up with a solution. The vast variety of problems which are **NP**-complete, but are not known to be in **P**, is considered a support to the **P** ≠ **NP** conjecture. A language is **NP**-*complete* if it is in **NP** and every language in **NP** is polynomially-reducible to it (see definition below). Hence, if some **NP**-complete language is in **P** then **NP** = **P**.

Definition: A language L is *polynomially-reducible* to the language L' if there exist a deterministic polynomial-time Turing machine M so that $x \in L$ if and only if $M(x) \in L'$.

Traditionally in Computer Science, NP-completeness proofs for languages are considered evidence to the intractability of the corresponding decision problem, since if **P** ≠ **NP** then membership in no NP-complete language can be determined in (deterministic) polynomial-time. Among the languages shown to be NP-Complete are *Satisfiability* (of propositional formulae), the *Traveling Salesman Problem* as well as many other optimization problems, and *Graph Colourability* as well as many other combinatorial problems.

2.3. Probabilistic Polynomial-Time Computations

Recent treads in computer science regard random computing steps as feasible. Following this approach, we consider computations which can be carried out by *Probabilistic* polynomial-time Turing machines as modeling efficient computations. A probabilistic Turing machine is an "extended" Turing machine that (based on its local configuration) chooses its next move at random (with uniform probability distribution) among a finite number of possibilities. (In a deterministic Turing machine, the next move is determined by the local configuration.) Without loss of generality, we assume that the number of possibilities (for the next local configuration) is either 1 or 2. One can then view the machine as tossing an unbiased coin before each

move and determining the next move using the outcome of the coin. On input x, the output of a probabilistic Turing machine M is a random variable defined over the probability space of all possible internal coin tosses. Equivalently, probabilistic Turing machines can be viewed as deterministic machines with two inputs: the ordinary input, and an auxiliary "random input". One then considers the probability distributions defined by fixing the first input and letting the auxiliary input assume all possible values with equal probability.

In particular, the complexity class **BPP** (*BPP* stands for **B**ounded-away-error **P**robabilistic **P**olynomial-time) is defined as the set of languages such that for every $L \in BPP$ there exists a probabilistic polynomial-time Turing machine M satisfying the following two conditions:

1) $Prob(M(x)=1) > \frac{2}{3}$ if $x \in L$.

2) $Prob(M(x)=0) > \frac{2}{3}$ if $x \notin L$.

It should be stressed that this definition is robust under substitution of $\frac{2}{3}$ by either $\frac{1}{2}+p(|x|)$ or $1-2^{-p(|x|)}$, where $p(\cdot)$ is an arbitrary positive polynomial. The following thesis captures the association of "efficient computation" with probabilistic polynomial-time computations.

Thesis: *BPP correspond to the set of problems which can be solved "efficiently".*

2.4. One-way Functions

It is generally believed that **P** $\neq$ **NP**, and furthermore that **BPP** $\neq$ **NP**. However, this does not necessarily mean that coming-up with hard instances of a "hard" language is easy. (Such instances exist, but may be hard to find.) The reader should note that both **NP** and **BPP** consider the worst-case complexity of problems, not their average-case complexity. For the results in this article we need a stronger assumption than **NP** $\neq$ **BPP**: namely that there are problems which are hard on most (or at least on a "non-negligible" portion) of the instances. This is formulated in terms of the infeasibility of inverting functions, which are easy to evaluate (in the forward direction).

Definition 1: A function $f:\{0,1\}^* \to \{0,1\}^*$ is called *one-way* if the following two conditions hold:

1) There exist a (deterministic) polynomial-time Turing machine that on input x outputs $f(x)$.

2) For **any** probabilistic polynomial-time Turing machine M', any constant $c > 0$, and sufficiently large n

$$Prob\left(M'(f(x),1^n) \in f^{-1}(f(x)) \right) < \frac{1}{n^c},$$

where the probability is taken over all x's of length n and the internal coin tosses of M', with uniform probability distribution.

Remark: The role of 1^k in the above definition is to allow machine M to run in time polynomial in the length of the preimage it is supposed to find. (Otherwise, any function which shrinks the input more than by a polynomial amount will

be considered one-way.)

Motivation to the notion of a negligible fraction: In the definition above, we have required that any machine trying to invert the function will succeed only on a "negligible" fraction of the inputs. We call *negligible* any function $\mu(\cdot)$ that remains smaller than 1 when multiplied by any polynomial We call *negligible* any function $\mu(\cdot)$ that remains smaller than 1 when multiplied by any polynomial (i.e. for every polynomial $p(\cdot)$ the limit of $\mu(n)\cdot p(n)$, when n grows to infinity, is 0). We ignore events which occur with negligible probability (as a function of the input length) since they are unlikely to occur even when repeating the experiment polynomially many times. On the other hand, events which occur with non-negligible probability will occur with almost certainty when repeating the experiment for a reasonable (i.e. polynomial) number of times. Thus, our notion of an "experiment" with a negligible success probability is robust (under polynomial number of repetitions of the experiment).

Motivation for considering infinitely many input lengths: The notion of a polynomial-time Turing machine is meaningful only when considering infinitely many input lengths. (Otherwise one can always choose a polynomial which bounds the running time of a machine that halts on all inputs in some finite set.) Furthermore, for any instance length l, there exists a Turing machine M_l which successfully inverts the function on all instances x of length l within $|x|+|f(x)|$ steps (machine M_l just incorporates in its transition function the inverses for all instances in this finite set). The same happens whenever we consider the inversion task for a finite set of instance lengths. Both technical difficulties are resolved when considering an infinite set of input lengths.

Assumption: *There exist one-way functions.* Furthermore, there exist one-way 1-1 functions.

The following three number theoretic 1-1 functions are widely believed to be one-way :

Ex1) **Modular Exponentiation**: In particular, let p be a prime and g be a primitive element of Z_p^* (the multiplicative group modulo p). Define $ME(p,g,x)=(p,g,y)$, where y is the result of reducing g^x modulo p. Inverting ME is known as the *Discrete Logarithm Problem*.

Ex2) **RSA**: Let p and q be primes, $N=p\cdot q$ and e be relatively prime to $\phi(N)=(p-1)\cdot(q-1)$. Define $RSA(N,e,x)=(N,e,y)$, where y equals x^e mod N.

Ex3) **Modular Squaring**: In particular, let p and q be primes both congruent to 3 mod 4, and $N=p\cdot q$. Define $MS(N,x)=(N,y)$, where y equals x^2 mod N. (To make this function one-to-one, restrict x to be a quadratic residue modulo N.) Inverting $MS(N,\cdot)$ is computationally equivalent to factoring N; that is, the problems are reducible to one another through probabilistic polynomial-time transformations.

Remark: The requirement, in condition 2 of Definition 1, that the inverting machine succeeds only a negiligible fraction of the instances of that particular length can be relaxed to requiring that the machine fails on some non-negligible fraction. Namely,

(2′) There exists a constant $c>0$ such that for **any** probabilistic polynomial-time Turing machine M' and sufficiently large n

$$Prob\left[M'(f(x),1^n) \notin f^{-1}(f(x)) \right] > n^{-c},$$

where the probability is taken as above.

The relaxed form is equivalent to the original Definition 1, both with respect to the existence of arbitrary one-way functions (one-way 1-1 functions), and with respect to the above three particular functions being one-way.

3. RANDOMNESS

In this section, we survey a recent behaviouristic approach to randomness. In this approach a probability distribution is considered "pseudorandom" if no "efficient procedure" can distinguish it from the uniform probability distribution. Remarkably, pseudorandomness so defined is expandable in the sense that (assuming the existence of 1-1 one-way functions) short pseudorandom sequences can be deterministically and efficiently expanded into much longer pseudorandom sequences.

3.1. Definition of Pseudorandom Distributions

A key definition in this approach is that of the infeasibility of distinguishing between two probability distributions. This behaviouristic definition, views distributions as equal if they cannot be told apart by any probabilistic polynomial-time test. Such a *test* receives as input a single string and outputs some statistics of the input. With no loss of generality, we may assume that the test outputs a single bit, which may be interpreted as a guess of the distribution from which the input was chosen. One considers the probability that, on input taken from the first distribution (resp. second distribution), the test outputs 1. If these two probabilities only differ by a negligible amount then the corresponding distributions are regarded as indistinguishable by this test.

Preliminaries (Probability Ensembles): A *probability distribution* is a function, π, from strings to non-negative reals such that $\sum_{\alpha \in \{0,1\}^*} \pi(\alpha) = 1$. A *probability ensemble* indexed by I is a sequence, $\Pi = \{\pi_i\}_{i \in I}$, of probability distributions. Throughout the entire article, we adopt the convention that the probability distributions in an ensemble assign non-zero probability only to strings of length polynomial in the length of the index of the distribution.

Motivation to defining ensembles: Probability ensembles are defined so that we can consider the asymptotic behaviour of arbitrary polynomial-time Turing machines on inputs taken from a probability distribution.

Definition 2 (Polynomial Indistinguishability - finite version): Let $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ and $\Pi_2 = \{\pi_{2,i}\}_{i \in I}$ be two probability ensembles each indexed by elements of I. Let T be a probabilistic polynomial-time Turing machine (hereafter called a *test*). The test gets two inputs: an index i and a string α. Denote by $p_1^T(i)$ the probability that, on input index i and a string α chosen according to the distribution $\pi_{1,i}$, the test T outputs 1 (i.e., $p_1^T(i) = \sum_{\alpha} \pi_{1,i}(\alpha) \cdot Prob(T(i,\alpha)=1)$). Similarly, $p_2^T(i)$ denotes the probability that, on input i and a string chosen according to the distribution $\pi_{2,i}$, the test T outputs 1. We say that Π_1 and Π_2 are *polynomially indistinguishable* if for all probabilistic polynomial-time tests T, all constants $c > 0$ and all sufficiently large $i \in I$

$$|p_1^T(i) - p_2^T(i)| < |i|^{-c}.$$

Motivation to having the index as an auxiliary input to the test: In the above definition, giving the index as auxiliary input to the test is not essential. However, in subsequent definitions presented in this article this convention plays an important technical role. For the sake of uniformity of definitions, we adopt this convention all along.

A probabilistic interpretation of Definition 2: Two probability distributions π_1 and π_2 are equal if they assign identical probability mass to the same string (i.e. if for all $\alpha \in \{0,1\}^*$, $\pi_1(\alpha)=\pi_2(\alpha)$). Two distributions are *statistically close* if they assign "about the same" mass to the same subsets of strings (i.e. if for all $S \subseteq \{0,1\}^*$, the difference between the sums $\sum_{\alpha \in S} \pi_1(\alpha)$ and $\sum_{\alpha \in S} \pi_2(\alpha)$ is negligible). Loosely speaking, two distributions are polynomial indistinguishable if they assign "about the same" probability mass to any efficiently recognizable set of strings.

An important special case of indistinguishable ensembles is that of probability ensembles which are polynomially indistinguishable from a uniform probability emsemble. These ensembles are called pseudorandom since, for all practical purposes, they are as good as truly unbiased coin tosses. This is clearly a behaviouristic point of view.

Definition 3 (Pseudorandom Distributions - finite version): Let $l:\{0,1\}^* \to N$ be a (*length*) function (mapping strings to integers), $\pi^l_{0,i}$ denote the uniform probability distribution on the set $\{0,1\}^{l(i)}$, and $\Pi^l_0 = \{\pi^l_{0,i}\}_{i \in I}$. Let $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ be a probability ensemble indexed by I. We say that Π_1 is *pseudorandom* if it is polynomially indistinguishable from π^l_0, for some length function l.

The above definitions (2 and 3) are titled "finite version" because each of the probability distributions (in the ensembles considered) is a function from *finite* strings to non-negative reals. The infinite version of these definitions considers instead distributions on *infinite* bit-sequences (infinite strings). For the infinite versions, we need to modify the definition of polynomially indistinguishable distributions, so that the tests run in time polynomial in the length of the first input (the index), while the second input may be infinite. (Here, the first input is essential in order to define the running time of the test!) Another modification, is in defining $\pi_{0,i}$ as a uniform distribution over the set of infinite strings. The technical details are omitted.

3.2. On the Expandability of Pseudorandom Distributions

Having defined pseudorandom ensembles, it is natural to ask whether such ensembles do exist. The answer is trivially affirmative, since the uniform ensemble is pseudorandom (being indistinguishable from itself!). However, this answer is of no interest. We would like to know whether ensembles which are not uniform, and furthermore are not statistically close to uniform, can be pseudorandom. Furthermore, can such ensembles be constructed using less coin tosses than the length of the strings in their support [2]? As we shall see in this section, assuming the existence of one-way 1-1 functions, the answer to both questions is affirmative. A key definition capturing the second question follows.

Definition 4 (Pseudorandom Generator): Let $p(\cdot)$ be a polynomial satisfying $p(n) \geq n+1$. Let G be a deterministic polynomial-time Turing machine that on input any n-bit string, outputs a string of length $p(n)$. Let $\underline{n}$ denote the unary encoding of the integer n. We say that G is a *pseudorandom generator* if the probability ensemble defined by it is pseudorandom. Here, the ensemble defined by G is $\{G_{\underline{n}}\}$ where a string y has

2) The *support* of a probability distribution is the set of elements which are assigned non-zero probability.

probability $m \cdot 2^{-n}$ in the distribution $G_{\underline{n}}$ if there are exactly m strings of length n such that feeding each of them to G yields the output y.

Motivation to the unary encoding of the length: The length of the seed to G (i.e. n) is encoded in unary so that the strings in the support of $G_{\underline{n}}$ have length polynomial in n ($= |\underline{n}|$).

Feeding a pseudorandom generators with seeds taken from a uniform distribution (over $\{0,1\}^n$), yields a pseudorandom distribution. The following Theorem states that feeding a pseudorandom generator with seeds taken from a pseudorandom distribution yields a pseudorandom distribution over longer strings.

Theorem 1: Suppose that $\Pi_1=\{\pi_{1,i}\}_{i \in I}$ is a pseudorandom ensemble, and G is a pseudorandom generator. Then the ensemble $\Pi_2=\{\pi_{2,i}\}_{i \in I}$, where $\pi_{2,i}$ is defined by feeding G with inputs according to the distribution $\pi_{1,i}$, is also pseudorandom.

Proof's Idea: Assume to the contrary that there exists a (polynomial-time) test T distinguishing between Π_2 and the uniform distribution. Then at least one of the following two statements hold:

1) The test T also distinguishes $\{G_{\underline{n}}\}$ from the uniform distribution (in contradiction to G being a pseudorandom generator).
2) The test T can be modified into a test T' (which first applies G to the tested string and then runs T on the result) so that T' distinguish Π_1 from the uniform distribution (thus contradicting the hypothesis that Π_1 is pseudorandom). □

We return to the fundamental question of *whether pseudorandom generators do exist.* We will see that, under the assumption that one-way 1-1 functions exist, the answer is **yes**. The following definitions and results are used in order to prove this implication. In particular, the equivalence of Definition 3 and Definition 5 plays an important role in proving the pseudorandomness of the construction presented below. Definition 5 concerns the feasibility of predicting the next bit in a string, which is taken from some distribution. The predictor is given only a prefix of the string. The question is whether there exists an efficient predictor which succeeds with probability non-negligibly greater than ½.

Definition 5 (Unpredictability): Let $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ be a probability ensemble indexed by I. Let M be a probabilistic polynomial-time Turing machine that on inputs i and y outputs a single bit (called the *guess*). Let $bit(\alpha,r)$ denote the r-th bit of the string α, and $pref(\alpha,r)$ denote the prefix consisting of the first r bits of the string α (i.e. $pref(\alpha,r) = bit(\alpha,1)bit(\alpha,2)\cdots bit(\alpha,r)$). We say that the machine M *predicts the next bit* of Π_1 if for some $c>0$ and infinitely many i's

$$Prob\left(M(i, pref(\alpha,r)) = bit(\alpha,r+1)\right) \geq \frac{1}{2} + |i|^{-c},$$

where the probability space is that of the string α chosen according to $p_{1,i}$, the integer r chosen at random with uniform distribution in $\{0,1,...,|\alpha|-1\}$ and the internal coin tosses of M. We say that Π_1 is *unpredictable* if there exist **no** probabilistic polynomial-time machine M which predicts the next bit of Π_1.

Definition 5 can be viewed as a special case of Definition 3. Any predictor can be easily converted into a test which outputs 1 if and only if the guess of the predictor is correct. The resulting test will distinguish an ensemble from the uniform ensemble if and only if the original predictor's guesses are non-negligibly better than "random". Interestingly, the special case is not less powerful. Namely, each successful distinguisher can be converted into a successful predictor.

Theorem 2: *Let* Π_1 *be a probability ensemble. Then* Π_1 *is pseudorandom if and only if it is unpredictable.*

Proof's Idea: Assume that T is a test which distinguishes $\pi_{1,i}$ from the uniform distribution. We consider the behaviour of T when fed with strings taken from the *hybrid* distributions $H_i^{(j)}$, where $H_i^{(j)}$ is defined as the distribution resulting by taking the first j bits of a string chosen from $\pi_{1,i}$ and letting the other bits be uniformly distributed. There must be two adjacent hybrids, $H_i^{(j)}$ and $H_i^{(j+1)}$, which are distinguishable by T. The $j+1$st bit is predicted using this "gap". □

The notion of a hard-core predicate (presented below) plays a central role in the construction of pseudorandom generators. Intuitively, a hard-core of a function f is a predicate ($b(x)$) which is easy to evaluate (on input x) but hard to even approximate when given the value of the function ($f(x)$). Recall that f is one-way if it is easy to evaluate (i.e. compute $f(x)$ from x) but hard to invert (i.e. compute x from $f(x)$). Thus, the hard-core maintains in a strong sense both the easyness (in the forward direction) and the hardness (in the backward direction) of the function.

Definition 6 (Hard-core Predicate): Let $f:\{0,1\}^* \to \{0,1\}^*$ and $b:\{0,1\}^* \to \{0,1\}$. The predicate b is said to be *a hard-core* of the function f if the following two conditions hold

1) There is a deterministic polynomial-time Turing machine that on input x returns $b(x)$.
2) There is **no** probabilistic polynomial-time Turing machine M' such that for some $c>0$ and infinitely many n

$$Prob\left[M'(f(x)) = b(x) \right] \geq 1/2 + n^{-c},$$

where the probability is taken over all possible choices of $x \in \{0,1\}^n$ and the internal coin tosses of M' with uniform probability distribution.

Clearly, if the predicate b is a hard-core of the 1-1 function f then f is hard to invert. Assuming that either of the functions presented in subsection 2.4 is one-way, predicates which constitutes corresponding hard-core can be presented. For example, the least significant bit is a hard-core of *RSA* (i.e., given $RSA(N,e,x)$ one cannot efficiently predict the least significant bit of x). In fact, every one-way function f can be "transformed" into a one-way function f' with a corresponding hard-core predicate b'. Thus, unpredictability and computational difficulty play dual roles.

Theorem 3: *If there exist one-way functions (resp. one-way 1-1 functions) then there exist one-way functions (resp. one-way 1-1 functions) with a hard-core predicate.*

Proof's Idea: The proof uses the observation that if f is one-way then there must be a bit in its argument x that cannot be efficiently predicted from $f(x)$ with success probability greater than $1-1/|x|$. (Otherwise, with constant probability, all the bits of the argument can be predicted correctly and the argument can be retrieved.) Let $b(i,x)$ denote the ith bit of x. For $|x_1| = |x_2| = \cdots = |x_{n^3}| = n$, define

$$f'(x_1,x_2,...,x_{n^3}) = f(x_1)f(x_2)\cdots f(x_{n^3}),$$

$$b'(x_1,x_2,...,x_{n^3}) = \left(\sum_{i=1}^{n} \sum_{j=1}^{n^2} b(i,x_{(i-1)\cdot n^2+j}) \bmod 2 \right).$$

It can be shown that the predicate b' is a hard-core of f'. The proof *does not* reduce to showing that a (sufficiently long) sequence of biased and independent 0-1 random variables has sum mod 2 which is almost unbiased (since the prediction errors on the various predicates are not random variables)! □

Having a one-way 1-1 function with a hard-core predicate suffices for the following construction of pseudorandom generators.

Construction 1: Let f be a one-way 1-1 function and b a hard-core predicate of f. We define the following polynomial-time Turing machine G. On input x, machine G computes the bits $b_i = b(f^{(i)}(x))$, where $1 \le i \le 2|x|$ and $f^{(i)}$ denotes the function f iteratively applied i times. Machine G outputs $b_{2|x|}\cdots b_2 b_1$.

Lemma 1: Let f, b and G be as in Construction 1. Then $\{G_{\underline{n}}\}$ defined as in Definition 4 is unpredictable.

Proof's Idea: An efficient predictor of the sequence defined above can be easily converted into a machine M that on input $f(x)$ guesses $b(x)$ with success probability greater than $1/2$. On input $f(x)$, machine M computes the sequence $b(f^{(k)}(x)),...,b(f^{(2)}(x)),b(f(x))$ and obtains a prediction for $b(x)$. □

Combining Theorem 3, Lemma 1 and Theorem 2, we get

Theorem 4: *If there exist one-way 1-1 functions then there exist pseudorandom generators.*

3.3. Discussion

Before presenting further extensions and applications of the above approach to randomness, let us discuss several conceptual aspects.

Behavioristic versus Ontologic

The behaviouristic nature of the above approach to randomness is best demonstrated by confronting this approach with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string which does not have a substantially simpler (i.e. shorter) explanation than itself. Considering the simplest explanation of a phenomenon is certainly an ontologic approach. In contrast, considering the effect of phenomena on certain objects, as underlying the definition of pseudorandomness (above), is a behaviouristic approach.

Furthermore, assuming the existence of one-way 1-1 functions, there exist probability distributions which are not uniform (and are not even statistically close to a uniform distribution) that nevertheless are indistinguishable from a uniform distribution (by any efficient method). Thus, distributions which are ontologically very different, are considered equivalent by the behaviouristic point of view taken in the definitions above.

A Relativistic View of Randomness

Pseudorandomness is defined above in terms of its observer. It is a distribution which cannot be told apart from a uniform distribution by any efficient (i.e. polynomial-time) observer. Thus, pseudorandomness is subjective to the abilities of the observer. To illustrate this point consider the following *mental experiment*.

> Alice and Bob want to play "head or tail" in one of the following four ways. In all of them Alice flips an unbiased coin and Bob is asked to guess its outcome before the coin rests on the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first way, Bob has to announce his guess before Alice flips the coin. Clearly, in this way Bob wins with probability 1/2. In the second way, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2. The third way is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth way, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve his guess of the outcome of the coin substantially.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Pseudorandom ensembles are unpredictable by probabilistic polynomial-time machines (associated with feasible computations), but may be predictable by infinitely powerful machines (not at our disposal!).

Effectiveness and Applicability

Another interesting property of the above approach to randomness is that it is effective in the following two senses: First, one may construct an efficient (universal) test that distinguishes pseudorandom distributions from ones which are not pseudorandom. In contrast, the problem of determining whether a string is Kolmogorov-random is undecidable. Second, assuming the existence of one-way 1-1 functions, long pseudorandom strings can be efficiently and deterministically generated from much shorter pseudorandom strings. Clearly, this cannot be the case with Kolmogorov-random strings.

The existence of pseudorandom generators has applications to the construction of efficient probabilistic algorithms (Turing machines). Such algorithms maintain the same performance when substituting their internal coin tosses by pseudorandom sequences. Thus, for every constant $\varepsilon>0$, the number of truly random bits required in a polynomial-time computation on input x can be decreased (from $poly(|x|)$ to $|x|^{\varepsilon}$.

Randomness and Computational Difficulty

Randomness and computational difficulty play dual roles. This was pointed out already when discussing one-way functions and hard-core predicates. The relationship between pseudorandom generators and one-way computations is even a better illustration of this point. We have shown above that the existence of one-way 1-1 functions implies the existence of pseudorandom generators. On the other hand, one can readily verify that any pseudorandom generator constitutes a one-way function.

3.4. Further Extension: Pseudorandom Functions or Experimenting with the Random Source

In the previous subsection we have (implicitly) modelled phenomena as single events (bit strings). This model suffices for describing phenomena in which the observer is passive: he can only record the events which occur. A more powerful model allows the observer to conduct experiments. Namely, "feed" the phenomenon with some values and measure the events which correspond to these values. Modelling a phenomenon as a function from events to events (or as a function from environment values to actions) is thus natural and useful. As in the previous subsections, we will present definitions for a pair of indistinguishable phenomena, a pseudorandom phenomenon and a generator of the latter. In other words, we will present definitions for indistinguishability of functions, pseudorandom functions, and pseudorandom function generators.

For our definition of indistinguishable function ensembles we consider Turing machines with oracles. These machines are able, in addition to the traditional computing steps, to make *oracle queries*: place a string on a special tape and read an "answer" in the next step. Loosely speaking, we will say that two function ensembles are indistinguishable if any polynomial-time oracle Turing machine cannot distinguish the case that its oracle is a function taken from the first ensemble and the case that the oracle is a function taken from the second.

Definition 7 (Indistinguishability of Functions, Pseudorandom Functions and Function Generators):
Let $F_1=\{F_{1,i}\}_{i\in I}$ and $F_2=\{F_{2,i}\}_{i\in I}$ be two function ensembles, where $F_{j,i}$ is a probability distribution on the functions $f:\{0,1\}^{|i|}\to\{0,1\}$. We say that F_1 and F_2 are *polynomially indistinguishable* if for every probabilistic polynomial-time oracle machine M, every constant $c>0$ and all sufficiently large $i\in I$

$$|p_1^M(i)-p_2^M(i)| < |i|^{-c},$$

where $p_j^M(i)$ is the probability that M outputs 1 on input i when querying an oracle randomly chosen from the distribution $F_{j,i}$.
The function ensemble $F=\{F_i\}_{i\in I}$ is *pseudorandom* if it is polynomially indistinguishable from the ensemble $H=\{H_i\}_{i\in I}$, where H_i is the uniform probability distribution on the set of functions $f:\{0,1\}^{|i|}\to\{0,1\}$.
We say that $F=\{F_{\underline{n}}\}$ is a *pseudorandom function generator* if the following three conditions hold:

1) There exists a probabilistic polynomial-time machine M_1 that, on input $\underline{n}$, randomly selects a function f from the distribution $F_{\underline{n}}$, and outputs a (succinct) description of f (denoted $\tilde{f}$).

2) There exists a (deterministic) polynomial-time machine M_2 that, on input $\tilde{f}$ (a description of $f:\{0,1\}^n \rightarrow \{0,1\}$) and a string x ($\in \{0,1\}^n$), outputs $f(x)$. That is, $M_2(\tilde{f},x) = f(x)$.

3) The ensemble F is pseudorandom.

Similar definitions apply to function ensembles consisting of distributions F_i on functions $f:\{0,1\}^{|i|} \rightarrow \{0,1\}^{|i|}$. Furthermore, one can easily transform ensembles of the first kind to ones of the second type, and vice versa.

As in subsection 3.2, we now ask whether there exist non-trivial ensembles of pseudorandom functions, and furthermore whether such ensembles can be efficiently generated. It turns out that this question reduces to the question handled in subsection 3.2. Namely,

Theorem 5: *Pseudorandom function generators exist if and only if pseudorandom generators exist.*

Proof's Idea: The "only if" direction of Theorem 5 is easy. The generator first uses M_1 to get an $\tilde{f}$ and next uses M_2 to evaluate $f(1)$, $f(2)$,... The "if" direction of the Theorem also has a constructive proof. The construction proceeds in two steps: First one uses an arbitrary pseudorandom generator to construct a pseudorandom generator G that doubles the length of its input. Next, G is used to construct a pseudorandom function in the following manner. Let $G_0(x)$ denote the first $|x|$ bits output by G on input x, and $G_1(x)$ denote the last $|x|$ bits output by G on input x. Extend the above notation so that for every bit σ and bit string α, $G_{\alpha\sigma}(x) = G_\alpha(G_\sigma(x))$. Now, let $f_x(y) = G_y(x)$, and $F_{\underline{n}}$ is the distribution obtained by picking x uniformly among all n bit strings and using the resulting function f_x. It can be shown that F so defined is a pseudorandom function generator. It is interesting to note that this is not the case if we let $f_x(y) = G_x(y)$. □

Further Discussion

It is interesting to point out the analogy between the above definition of pseudorandom functions and Turing's famous "test of intelligence". (In Turing's test of intelligence, one is interacting arbitrarily with an unknown entity which is either a human or a machine. The machine is said to be (pseudo)intelligent if the tester cannot distinguish the two cases.) In both settings one interacts with an unknown function in order to latter determine the "nature" of this function. Failure to determine the "true nature" is interpreted as a proof that the difference in nature is of no importance (as far as functionality goes...).

Pseudorandom functions can not be predicted, even not in the following weak sense: any probabilistic polynomial-time oracle Turing machine cannot predict the value of the oracle on an unasked query better than 50-50, when the oracle is a pseudorandom function. This resembles the following quotation of Turing:

> *I have set up on a Manchester computer a small programme using only 1000 units of storage, whereby the machine supplied with one sixteen figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values.*

3.5. Applications to Cryptography

The most obvious application of pseudorandomness to cryptography is making *one-time pads* a feasible and secure encryption method. One-time pads are the simplest and safest private-key cryptosystem. A cleartext is encrypted by XORing [3] its bits with the currently initial segment of the (*randomly selected*) key, and the resulting ciphertext is decrypted by XORing its bits with the very segment of the key. Each segment of the key is deleted after use, and thus no information about the cleartext can be extracted from the ciphertext. The drawback of one-time pads is that the length of the key in use must equals or even exceed the length of the messages sent. Namely, in order to *secretly* pass a message of length l one must exchange secretly another message of length l. This is not satisfactory both from a theoretical and practical point of view, since the aim is to achieve high level of security in a much lower "cost". In practice, "pseudorandom sequences" are used instead of the randomly selected key of the one-time pad, *but security can no longer be asserted*. Assuming the existence of pseudorandom bit generators (in the sense discussed in section 3.2), one can replace the key of the one-way pad by a pseudorandom sequence and **prove** that the resulting cryptosystem is secure in the following sense: *whatever can be efficiently computed from the ciphertext can be efficiently computed without it.* In other words, as far as polynomial-time computations are concerned, no information about the cleartext is *revealed* from the ciphertext.

Other applications of pseudorandomness to Cryptography use the construction of pseudorandom functions (Theorem 5, section 3.4). For example, it is possible to produce unforgeable *message authentication tags* and *time-stamps*. Assume two parties A and B, sharing a secret key, communicate over a channel tampered by an adversary C. The adversary may inject messages on the channel. The parties would like to be able to verify that a message has arrived from their counterpart, and not from the adversary. It is suggested that in order to authenticate a message M, party A just applies the pseudorandom function f to M, and sends $f(M)$ as the authentication tag of M. Party B may then verify the validity of this authentication tag, being confident that the message has been sent by A (and not injected by C). We stress that if f is a pseudorandom function then the above scheme is **provably** secure in the following sense: *even if C gets polynomially many authentication tags to messages of his choice he cannot produce in polynomial-time an authentication tag to any other message.*

3) XORing two bit strings means applying exclusive-or (XOR) to each pair of corresponding bits.

4. INTERACTIVE PROOFS

In this section we survey a recent behaviouristic approach to the notion of an (efficiently verifiable) proof. In this approach, a proof system for proving membership in a language L is a two-party protocol for a *prover* and a *verifier* so that the prover can convince the verifier to accept x (with high probability) if and only if $x \in L$.

4.1. Definition of Interactive Proofs

Before defining the notion of an interactive proof, we define the notion of an interactive pair of Turing machines, which captures the intuitive notion of a two-party protocol.

Definition 8 (Pair of Interactive Turing machines): An *interactive Turing machine (ITM)* is a six-tape deterministic Turing machine with a read-only *input tape*, a read-only *random tape*, a read/write *work tape*, a read-only *communication tape*, a write-only *communication tape*, and a write-only *output tape*. The string which appears on the input tape is called the *input*. The contents of the random tape can be thought of as the outcomes of unbiased coin tosses. The string which appears on the output tape when the machine halts is called the *output*. The contents of the write-only communication tape can be thought of as messages sent by the machine; while the contents of the read-only communication tape can be thought of as messages received by the machine.

An *interactive pair of Turing machines* is a pair of ITMs which share their communication tapes so that the read-only (communication) tape of the first machine is the write-only (communication) tape of the second machine, and vice versa. Let M_1 and M_2 be an interacting pair of ITMs, then $[M_2(x_2),M_1(x_1)]$ denotes the output of M_1 on input x_1, when M_2 has input x_2.

Intuitively, an interactive proof system for a language L is a two-party protocol for a "powerful" *prover* and a probabilistic polynomial-time *verifier* satisfying the following two conditions with respect to the common input, denoted x. If $x \in L$ then with very high probability the verifier is "convinced" of this fact, when interacting with the prover. If $x \notin L$ then no matter what the prover does, he cannot fool the verifier (into believing that "x is in L"), except for with very low probability. The first condition is referred to as the *completeness* condition, while the second condition is referred to as *soundness*

Definition 9 (Interactive Proof): An *interactive proof for a language L* is a pair of ITMs P and V satisfying the following conditions:

0) On input x machine V make at most $p(|x|)$ steps, where $p(\cdot)$ is a fixed polynomial.

1) *Completeness*: For every constant $c>0$, and all sufficiently long $x \in L$

$$Prob\left([P(x),V(x)]=1\right) \geq 1-|x|^{-c}.$$

2) *Soundness*: For every constant $c>0$, every ITM P', all sufficiently long $x \notin L$, and every $y \in \{0,1\}^*$,

$$Prob\left([P'(y),V(x)]=0\right) \geq 1-|x|^{-c}.$$

Denote by **IP** the class of languages having interactive proofs.

Remarks: Note that it does not suffice to require that the verifier cannot be fooled by the predetermined prover (such a mild condition would have presupposed that the "prover" is a trusted oracle). **NP** is a special case of interactive proofs, where the interaction is trivial and the verifier tosses no coins.

Example of an interactive proof

To illustrate the definition of an interactive proof we present an interactive proof for *Graph Non-Isomorphism*. The input is a pair of graphs G_1 and G_2, and one is required to prove that there exists no 1-1 edge-invariant mapping of the vertices of the first graph to the vertices of the second graph. (A mapping π from the vertices of G_1 to the vertices of G_2 is *edge-invariant* if the nodes v and u are adjacent in G_1 if and only if the nodes $\pi(v)$ and $\pi(u)$ are adjacent in G_2.) It is interesting to note that no short NP-proofs are known for this problem; namely Graph Non-Isomorphism is *not known* to be in **NP**.

The interactive proof proceeds as follows: The verifier chooses at random one of the two input graphs, say G_α ($\alpha \in \{1,2\}$). The verifier creates a random isomorphic copy of G_α and sends it to the prover, which is supposed to answer with $\beta \in \{1,2\}$. The verifier interprets $\beta=\alpha$ as evidence that the graphs are not isomorphic; while $\beta \neq \alpha$ leads him to reject. This is repeated several times (with independent random choices!) to collect stochastic evidence. The verifier accepts (the graphs as non-isomorphic) if and only if all of the provers responses are correct.

If the two graphs are not isomorphic, the prover has no difficulty to always answer correctly (i.e. a β equal to α), and thus the completeness condition is met. If the two graphs are isomorphic it is impossible to distinguish a random isomorphic copy of the first from a random isomorphic copy of the second, and the probability that the prover answers correctly to one "query" is at most 1/2. The probability that the prover answers correctly all t queries is $\leq 2^{-t}$ and the soundness condition is satisfied.

4.2. Discussion

The terminology of interactive proofs explicitly deals with the two fundamental computational tasks related to proof systems: producing a proof and verifying the validity of a proof. For many years **NP** was considered *the formulation* of "whatever can be efficiently verified". This stemmed from the traditional association of deterministic polynomial-time computation with efficient computation. The growing acceptability of probabilistic polynomial-time computations as reflecting efficient computations is the basis of the more recent formalization (namely **IP**) of "whatever can be efficiently verified". As we regard random computing steps as feasible, there is no reason not to allow the verifier to make such steps. Following our convention of disregarding events that occur with negligible probability, we disregard the probability of error in such proofs. (Objections to this approach are discussed in the sequel.) Also, there seem to be no reason to restrict the interaction between the prover and verifier, as long as the verifier remains efficient in term of (the length of) the claim to be proven. In fact, a hierarchy of complexity classes, parametrized by the number of messages exchanged between the prover and verifier, seems to emerge. Namely, **IP**($f(\cdot)$) denotes the class of languages

having interactive proofs in which up to $f(|x|)$ messages are exchanged on input x.

The definition of interactive proofs does not meet the traditional notion of a "proof", which disallows the possibility of error. The fact that in an interactive proof an error may occur only with an overwhelmingly small probability does not overrole the objection of a purist (namely that "this is not a proof"). But even the purist must agree that an interactive proof captures the intuitive notion of a convincing argument. Any reasonable person, trusting his own coin tosses, will believe statements "proved" through means of an interactive proof and ignore the overwhelmingly small possibility of an error. Ignoring errors which may occur with overwhelmingly small probability is clearly a behaviouristic approach to life. In other words, as far as all practical purposes are concerned, an interactive proof is as good as a "real proof" (i.e. an **NP** proof).

Another possible objection to interactive proofs is that they are not "transferrable", but rather convince only a party that either actively participates in them (as a verifier) or believes that the verifier's "random moves" were unpredictable by the prover. (Consider for example the interactive proof for Graph Non-Isomorphism: if the prover can predict the verifier's coin tosses then he can answer correctly even if the graphs are isomorphic.)

In going from **NP** to interactive proofs (i.e. **IP**) we gave away certainty and transferrability of the "proof". *Does this buy us anything?* Unless $\mathbf{IP} \subseteq \mathbf{BPP}$ (and hence $\mathbf{NP} \subseteq \mathbf{BPP}$), the answer is affirmative as interactive proofs allow us to introduce (non-trivially) the notion of zero-knowledge proofs (see next section). Another possible gain of interactive proofs is that they allow to prove membership in languages *not known* to be in **NP** (e.g. Graph Non-Isomorphism). (*Proving* that $\mathbf{IP} \neq \mathbf{NP}$ is way out of the "current state of the art" as it will imply that **NP** is strictly contained in **PSPACE**.)

An interesting question regarding interactive proofs is what ingredients or parameters determine their power. The most important discovery in this direction is that restricting the verifier to only send the prover the outcome of his coin tosses, does not decrease the power of the proof system. In other words, whatever can be proven with a verifier that cleaverly chooses his "questions", can be proven with a verifier that chooses his "questions" at random. (The above restricted type of interactive proof is called *an Arthur Merlin game*.) Another result is that increasing the number of interactions by a multiplicative factor does not increase the power of the system (i.e. for every constant $c>0$ and function f, $\mathbf{IP}(f(\cdot)) = \mathbf{IP}(c \cdot f(\cdot))$). In particular, an interaction in which the verifier gets an answer to one randomly chosen question is as powerful as an interaction consisting of a bounded sequence of questions and answers. Finally, one can show that the error probability in the completeness condition (of Definition 9) is not essential, while the error probability in the soundness condition is essential (unless $\mathbf{NP} = \mathbf{IP}$).

5. ZERO - KNOWLEDGE PROOFS

In this section, we survey the notion of zero-knowledge proofs. To examplify this notion consider the set (i.e. language) of satisfyable propositional formulae. An easy way to convince a polynomial-time verifier that a formula ψ is satisfiable is to demonstrate a truth assignment which satisfies ψ. This proof, however, reveals much more than the fact that ψ is satisfiable: it yields a satisfying assignment. The reader should note that, unless $\mathbf{P} \neq \mathbf{NP}$, a satisfying assignment is hard to get even if one knows that such an assignment does exist. Thus, the verifier in the above proof obtains (from the prover) knowledge which he could not compute by himself, even if he believed that ψ is satisfiable. In a zero-knowledge proof, the verifier will be convinced that ψ is satisfiable without getting a satisfying assignment nor obtaining any knowledge about the formula which is not attainable from the formula itself and the fact that it is satisfiable. Thus, the verifier in a zero-knowledge proof is essentially in the same situation as if he is told by a trusted oracle that ψ is satisfiable.

5.1. Definition of Zero-Knowledge Proofs

Intuitively, a zero-knowledge proof is a proof which yields nothing but its validity. This means that for all practical purposes, "whatever" can be done after interacting with a zero-knowledge prover, can be done when just believing that the assertion he claims is indeed valid. (In "whatever" we mean not only the computation of functions but also the generation of probability distributions.) Thus, zero-knowledge is a property of the predetermined prover. It is the robustness of the prover against attempts of the verifier to extract knowledge via interaction. Note that the verifier may deviate arbitrarily (but in polynomial-time) from the predetermined program. This is captured by the formulation sketched below.

Denote by $[P,V^*(x)]$ the probability distribution generated by a machine V^* which interacts with (the prover) P on input $x \in L$. We say that the proof system is *zero-knowledge* if for all probabilistic polynomial-time interactive machines V^*, there exists a probabilistic polynomial-time machine M_{V^*} that on input x produces a probability distribution $M_{V^*}(x)$ such that $\{M_{V^*}(x)\}_{x \in L}$ and $\{[P,V^*(x)]\}_{x \in L}$ are polynomially-indistinguishable. (We stress that M_{V^*} is an ordinary machine which does not interact with P or any other machine.)

As we argued in section 3, polynomially-indistinguishable probability distributions should be considered equal for all practical purposes. It follows that the polynomially-indistinguishabilty of $[P,V^*(x)]$ and $M_{V^*}(x)$ suffices for saying that nothing substantial is gained by interacting with the prover, except of course conviction in the validity of the assertion $x \in L$.

Example of a zero-knowledge proof

To illustrate the above definition we present an *zero-knowledge* interactive proof for *Graph Isomorphism* (see definition of the problem in subsection 4.1). Before doing so let us stress that the isomorphism between the two graphs does constitute a proof for the fact that they are isomorphic, but that this proof is unlikely to be zero-knowledge (as it will imply that the problem of finding an isomorphism between a pair of graphs is in **BPP**).

Our zero-knowledge proof proceeds as follows: The prover creates a random isomorphic copy of the first graph (G_1). The verifier chooses at random one of the two input graphs, and asks the prover to present an isomorphism between the chosen graph and the graph sent by the prover. If the prover fails to present such an isomorphism the verifier rejects. This is repeated several times (with independent random choices!) to collect stochastic evidence. The verifier accepts (the graphs as isomorphic) if and only if all of the provers responses are correct.

If the two graphs are isomorphic, and the prover knows an isomorphism between them, he has no difficulty to always answer correctly. If the two graphs are not isomorphic it is impossible to answer both possible questions of the verifier. If the verifier chooses his t questions randomly then the probability that the prover answers correctly all of them is $\leq 2^{-t}$.

Following is a very rough sketch of the argument that the above proof is indeed zero-knowledge. One can simulate the conversations between the prover and a verifier by selecting at random one of the two input graphs and creating a random isomorphic copy of it. If the verifier would have asked to see that isomorphism, we can supply it. Otherwise, we repeat the process with new independent random choices. The expected number of repetitions needed to simulate one round of prover-verifier conversation is 2. The distribution created by taking only the successful repetitions equals the distribution of the original prover-verifier conversations.

5.2. Discussion

The definition of zero-knowledge presupposes that probabilistic polynomial-time is for "free". Namely, a conversation between the prover and the verifier which can be simulated by the verifier himself in probabilistic polynomial-time contains no knowledge. Thus, implicitly, knowledge is regarded as the result of a computation which is infeasible for the verifier himself.

Randomness and interaction are essential to the non-triviality of the notion of zero-knowledge. It can be shown that zero-knowledge proofs in which the verifier either tosses no coins or asks no questions exist only for languages in **BPP**. Note that such zero-knowledge proofs are of no interest since every language in **BBP** has a trivial zero-knowledge proof in which the prover sends nothing to the verifier!

The definition of zero-knowledge seems somewhat paradoxical: these proofs yield no knowledge in the sense that they can be constructed by the verifier who believes the statement, and yet these proofs do convince him. The "paradox" is resolved by noting that it is not the text of the conversation that convinces the verifier, but rather the fact that this conversation was held "on line". When constructing such a conversation text by himself the verifier works "off line", concatenating parts of possible conversations while deleting other parts.

5.3. Proving any NP Statement in Zero-Knowledge

In providing zero-knowledge proofs to any language in **NP**, we use the notion of **NP**-completeness (see section 2.2). Namely, we provide a zero-knowledge proof for one **NP**-complete language (denoted L_0) and derive such proofs for any other L ($\in$ **NP**) language by using the polynomial reduction of L to L_0. In presenting a zero-knowledge proof for L_0, we use any secure encryption scheme (in the sense of Goldwasser and Micali [GM]). The existence of such schemes is guaranteed by our assumption that one-way 1-1 functions exist.

As the basis for the construction (i.e. the language L_0), we use the set of 3-colourability graphs. The language *Graph 3-Colourability* consists of the set of graphs, the vertices of which can be coloured using three colours such that no two adjacent vertices are assigned the same colour.

A Zero-Knowledge Proof for Graph 3-Colourability

The common input to the following protocol is a graph $G(V,E)$, where $V=\{1,2,...,n\}$ is the vertex set and $E \subseteq V \times V$ is the edge set. In the following protocol, the prover needs only to be a probabilistic polynomial-time machine which gets a proper 3-colouring of G as an auxiliary input. Let us denote this colouring by ϕ ($\phi: V \rightarrow \{1,2,3\}$). Let $m = |E|$.

The following four steps are executed m^2 times, each time using independent coin tosses.

1) The prover chooses a random permutation of the 3-colouring, encrypts it, and sends it to the verifier. More specifically, the prover chooses at random a permutation $\pi \in Sym(\{(1,2,3\})$, encrypts (separately) each element of the sequence $\pi(\phi(1)),\pi(\phi(2)),...,\pi(\phi(n))$, and sends the resulting encrypted sequence to the verifier.

2) The verifier chooses at random an edge $e \in E$ and sends it to the prover. (Intuitively, the verifier asks to examine the colouring of the endpoints of $e \in E$.)

3) If $e=(u,v) \in E$ then the prover reveals the colouring of u and v and "proves" that they correspond to their encryptions. If $e \notin E$ then the prover stops.

4) The verifier checks the "proof" provided in step (3). Also, the verifier checks that the colours revealed are consistent (i.e. $\pi(\phi(u)) \neq \pi(\phi(v))$, and $\pi(\phi(u)),\pi(\phi(v)) \in \{1,2,3\}$). If either condition is violated the verifier *rejects* and stops. Otherwise the verifier continues to the next iteration.

If the verifier has completed all m^2 iterations then it *accepts*.

The reader can easily verify the following facts: If the graph is 3-colourable and both prover and verifier follow the protocol then the verifier always accepts. If the graph is not 3-colourable and the verifier follows the protocol then no matter how the prover plays, the verifier will reject with probability at least $(1-m^{-1})^{m^2} = \exp(-m)$. Thus, the above protocol constitutes an interactive proof system for 3-colourability. Clearly, this protocol yields no knowledge to the specified verifier, since all he gets is a sequence of random pairs. The proof that the protocol is indeed zero-knowledge (with respect to *any* verifier) is much more complex, and is omitted. We get,

Proposition: If there exist one-way 1-1 functions then there exist a zero-knowledge interactive proof system for 3-colourability.

Zero-Knowledge Proofs for all languages in NP

For every language L in **NP**, there exist an efficient transformation of instances of the language L to instances of 3-colourability. This transformation is called a *reduction*, and is guaranteed by the fact that 3-colourability is **NP**-Complete. Incorporating the standard reductions into the protocol for graph 3-colourability, we get

Theorem 6: *If there exist one-way 1-1 functions then every NP language has a zero-knowledge interactive proof system. Furthermore, in this proof system the prover is a probabilistic polynomial-time Turing machine which gets an NP-proof as an auxiliary input.*

Using Theorem 6, one can prove that any language in **IP** has a zero-knowledge proof. Thus, "whatever is efficiently provable" is "efficiently provable in a zero-knowledge manner".

5.4. Applications to Cryptography

Theorem 6 (above) yields an extremely powerful tool for the design of cryptographic protocols: the *ability to prove any NP statement in a zero-knowledge manner*. To better understand the relevance of this tool, let us briefly discuss the setting in which cryptographic protocols arise.

A cryptographic protocol is a sequence of interactive programs to be executed by *non-trusting* parties. Each party has a local input unknown to the others, and hereafter referred to as *his secret*. Typically, the protocol specifies actions to be taken by each of the parties based on his secret and previous messages. The protocol designer is thus faced with the following problem: how can one party verify that his counterpart has computed the next message in accordance with the protocol? Verification is difficult since the verifier does not know and is not supposed to know the secret of the transmitter. Zero-knowledge proofs for all NP statements are the answer to this problem. The transmitter's claim to having computed his message according to the protocol is an NP statement (and furthermore, the transmitter knows an NP-proof to it). By Theorem 6, this NP claim can be proven without yielding any knowledge of the prover's secret!

The above suggests a powerful methodology for the design of cryptographic protocols. First design your protocol assuming that all parties will follow it properly. Next compile the protocol using zero-knowledge proofs to a protocol which maintains the correctness and privacy of the original protocol even when a minority of the parties display arbitrary adversarial behaviour. The details of the complier are beyond the scope of this survey.

6. CONCLUSIONS

The fact that pseudorandom generators and functions exist under a reasonable complexity theoretic assumption (i.e. the existence of one-way 1-1 functions), must be considered at least a plausibility argument. Thus, every reasoning overruling the existence of such generators must incorporate a demonstration that one-way 1-1 functions do not exist. The *possible existence* of pseudorandom generators does not allow us to consider "unbounded" random behaviour as necessarily arising from an "unbounded" source of randomness, since a pseudorandom generator may expand a "bounded" amount of randomness to an "unbounded" amount of pseudorandomness. Furthermore, the possible existence of pseudorandom functions implies that a small amount of randomness suffices in order to efficiently determine a random mapping from huge sets into huge sets.

Also under the same complexity theoretic assumption, the folklore belief that one *necessarily* gains extra insight into a theorem by seeing its proof - is seriously shakened. Zero-knowledge proofs are proofs which yield no such insight. Whatever one can efficiently learn from the proof, one can deduce as easily by believing that the theorem is valid.

All the above was discovered through a behaviouristic approach to computational notions such as randomness and proofs. We believe that a behaviouristic approach is justified when studying computing devices, as much as it is unjustified when studying "thinking beings".

ACKNOWLEDGEMENTS

First of all, I would like to thank two remarkable people who had a tremendous influence on my professional development. Shimon Even introduced me into theoretical computer science and closely guided me in my first steps. Silvio Micali led my way in the evolving foundations of cryptography and shared with me his efforts of further developing them. Next, I would like to thank my colleagues Benny Chor and Avi Wigderson for their indispensable contribution to our joint research, and for the excitement and pleasure I had when collaborating with them. Special thanks to Leonid Levin for many interesting discussions.

I grateful to Janos Makowsky and Rolf Herken for suggesting me to write this article and providing guidelines for it. Thanks to Hugo Krawczyk for carefully reading an earlier version of the manuscript, pointing out some errors, and suggesting several improvements. Finally, thanks to Dassi.

BIBLIOGRAPHIC NOTES

For background on Computational Complexity consult an appropriate textbook such as [HU, ch. 12-13] and [GJ].

The notion of one-way functions was first suggested in [DH], and the most famous candidate is due to [RSA]. A 1-1 function which is one-way, unless factoring is easy appears in [R]. Definition 1 (*one-way functions*), however, is a weaker form and is due to [Y1]. A special case of Definition 2 (*indistinguishability*) first appeared in [GM], the general case is from [Y1]. Definitions 3 and 4 (*pseudorandomness*) are due to [Y1], while Definition 5 (*unpredictability*) appears in [BM]. Theorem 2 (*equivalence of Def's 3 and 5*) is implicit in [Y1]. Definition 6 (*hard-core predicate*), Construction 1 (*pseudorandom generator based on a hard-core predicate*) and Lemma 1 appear in [BM]. Theorem 3 (*existence of hard-core predicates assuming one-way 1-1 functions*) is implicit in [Y1], where a sketch of the proof of Theorem 4 (*pseudorandom generator based on one-way 1-1 functions*) appears. A finer analysis, which uses a weaker (necessary and sufficient) condition, of Theorems 3 and 4 appears in [L]. Predicates which are hard-core of the particular number theoretic functions mentioned in section 2.4, appear in [BM] and [ACGS]. Definition 7 (*pseudorandom functions*) and Theorem 5 (*pseudorandom generators imply pseudorandom function generators*) appear in [GGM]. Further developments appear in [LR].

Two different definitions of interactive proofs appear in [GMR] and [B], respectively. In section 4, we have used the definition of [GMR]. The definition in [B] (*Arthur Merlin games*) is a special case of it. The definitions were proven to be equivalent in [GS]. The interactive proof for Graph Non-Isomorphism originates from [GMW1].

The definition of a zero-knowledge proof originates from [GMR], which contains also a more general definition of the "amount of knowledge contained in a proof". The zero-knowledge proof for Graph Isomorphism as well as Theorem 6 (*all NP languages have zero-knowledge proofs*) appears in [GMW1]. (For a definition of secure encryption functions see [GM].) The applications mentioned in section 5.4 appear in [GMW1, GMW2]. Further developments appear in [Y2] and [GMW2].

REFERENCES

[ACGS] Alexi, W., B. Chor, O. Goldreich and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As the Whole", *Proc. 25th IEEE Symp. on Foundation of Computer Science*, 1984, pp. 449-457, (to appear in *SIAM J. Computing*).

[B] Babai, L., "Trading Group Theory for Randomness", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 421-429.

[BM] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM Jour. on Computing*, Vol. 13, 1984, pp. 850-864.

[DH] Diffie, W., and M. E. Hellman, "New Directions in Cryptography", *IEEE transactions on Info. Theory*, IT-22 (Nov. 1976), pp. 644-654

[GJ] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *Jour. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807.

[GMW1] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *Proc. of 27th Symp. on Foundation of Computer Science*, 1986, pp. 174-187.

[GMW2] Goldreich, O., S. Micali, and A. Wigderson, "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority", *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 218-229.

[GM] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299.

[GMR] Goldwasser, S., S. Micali, and C. Rackoff, "Knowledge Complexity of Interactive Proofs", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 291-304, (to appear in *SIAM J. Computing*).

[GS] Goldwasser, S., and M. Sipser, "Arthur Merlin Games versus Interactive Proof Systems", *Proc. 18th ACM Symp. on Theory of Computing*, 1986, pp. 59-68.

[HU] Hopcroft, J.E., and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Co., 1979.

[L] Levin, L.A. "One-Way Function and Pseudorandom Generators", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 363-365.

[LR] Luby, M., and C. Rackoff, "Pseudo Random Permutation Generators and DES", *Proc. 18th ACM Symp. on Theory of Computing*, 1986, pp. 356-363.

[R] Rabin, M.O. "Digitalized Signatures and Public Key Functions as Intractable as Factoring", MIT/LCS/TR-212, 1979.

[RSA] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM*, Vol. 21, Feb. 1978, pp. 120-126

[Y1] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

[Y2] Yao, A.C., "How to Generate and Exchange Secrets", *Proc. of the 27th IEEE Symp. on Foundation of Computer Science*, 1986, pp. 162-167.